

Stochastic Kronecker Graph on Vertex-Centric BSP

Ernest Ryu^{*} and Sean Choi[†]

^{*}Institute for Computational and Mathematical Engineering, Stanford University

[†]Department of Computer Science, Stanford University

Abstract

Recently Stochastic Kronecker Graph (SKG), a network generation model, and vertex-centric BSP, a graph processing framework like Pregel, have attracted much attention in the network analysis community. Unfortunately the two are not very well-suited for each other and thus an implementation of SKG on vertex-centric BSP must either be done serially or in an unnatural manner.

In this paper, we present a new network generation model, which we call Poisson Stochastic Kronecker Graph (PSKG), that generate edges according to the Poisson distribution. The advantage of PSKG is that it is easily parallelizable on vertex-centric BSP, requires no communication between computational nodes, and yet retains all the desired properties of SKG.

1 Introduction

With the advent of massive real-world network data and the computation power to process them, network analysis is becoming a major topic of scientific research. As approaches to model real-world networks, Stochastic Kronecker Graph (SKG) [7] and its predecessor R-MAT [5] have attracted interest in the network analysis community due to its simplicity and its ability to capture many properties of real-world networks. As a programming model to process large graphs, vertex-centric BSP, such as Pregel [8], Apache Giraph [2], GPS [9], and Apache Hama [3], has become increasingly popular as an alternative to MapReduce and Hadoop, which are ill-suited to run massive scale graph algorithms [8].

The two, however, are not well-suited for each other. The obvious approach of parallelizing SKG,

which is to generate edges in parallel, is not “vertex-centric” in nature and therefore is unnatural to program and runs inefficiently in vertex-centric BSP.

Therefore we present a new network generation model, which we call Poisson Stochastic Kronecker Graph (PSKG), as an alternative. In this model, the out-degree of each vertex is determined by independent but non-identical Poisson random variables and the destination node of the edges are determined in a recursive manner similar to SKG.

The resulting algorithm, PSKG, is essentially equivalent to SKG and will therefore retain all the desired properties of it. Unlike SKG, however, PSKG is embarrassingly parallel in a vertex-centric manner and therefore is very well-suited for vertex-centric BSP.

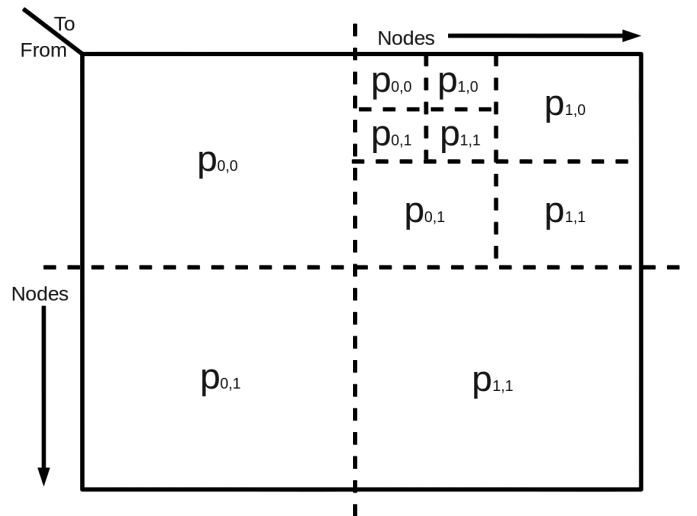


Figure 1: Illustration of SKG. At each of the k steps a sub-region is chosen with probability $p_{i,j}$.

Part of this work was done while the second author was visiting LinkedIn.

2 Theory and Algorithm

The main result of this paper, PSKG, is presented in Algorithm 3. We shall, however, start by discussing the original SKG and an equivalent formulation of it as this path will motivate PSKG. We then present the main algorithm. We conclude this section by discussing issues with load balancing.

Remark. *Throughout this paper we shall use zero-based indexing for vectors and matrices.*

2.1 Stochastic Kronecker Graph

Consider the problem of generating a random graph of E edges and $N = n^k$ vertices, where $k \in \mathbb{N}$. Let

$$P = \begin{bmatrix} p_{0,0} & \cdots & p_{0,n-1} \\ \vdots & \ddots & \vdots \\ p_{n-1,0} & \cdots & p_{n-1,n-1} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

be the “initiator matrix” where $p_{i,j} \geq 0$ and $\sum_{i,j} p_{i,j} = 1$.

The approach in SKG is to start off with an empty adjacency matrix and “drop” edges into the matrix one at a time. Each edge chooses one of the $n \times n$ partitions with probability $p_{i,j}$ respectively. The chosen partition is again subdivided into smaller partitions, and the procedure is repeated k times until we reach a single cell of the $N \times N$ adjacency matrix and place an edge. Figure 1 illustrates the idea and Algorithm 1 makes it concrete.

Algorithm 1 SKG

```

for  $i = 1, \dots, E$  do
   $u = v = 0$ 
  for  $j = 1, \dots, k$  do
    With probability  $p_{rs}$  choose subregion  $(r, s)$ 
     $u = nu + r; v = nv + s$ 
  end for
  Add edge  $(u, v)$ 
end for

```

Let $P_k = P \otimes P \otimes \cdots \otimes P$ be the k -th Kronecker power of P . Then we can interpret SKG as generating m edges independently¹ where any given edge is (u, v) with probability $(P_k)_{uv}$. Now we can apply Bayes’ rule.

$$\mathbf{P}(\text{edge is } (u, v)) = (P_k \mathbf{1})_u \frac{(P_k)_{uv}}{(P_k \mathbf{1})_v} \quad (1)$$

$$= \mathbf{P}(\text{destination is } u | \text{source is } v) \mathbf{P}(\text{source is } v | \text{destination is } u)$$

¹The edge generations are not quite independent due to possible “collisions.”

The decomposition (1) permits us to choose the source node first and then the destination node rather than simultaneously and we do this with a recursive algorithm to avoid the explicit construction of P_k . Let

$$U = \begin{bmatrix} \sum_{j=0}^{n-1} p_{0,j} \\ \vdots \\ \sum_{j=0}^{n-1} p_{n-1,j} \end{bmatrix} \quad V = \begin{bmatrix} \frac{p_{0,0}}{U_0} & \cdots & \frac{p_{0,n-1}}{U_0} \\ \vdots & \ddots & \vdots \\ \frac{p_{n-1,0}}{U_{n-1}} & \cdots & \frac{p_{n-1,n-1}}{U_{n-1}} \end{bmatrix}$$

and we arrive at Algorithm 2 which is equivalent to the original formulation of SKG.

Algorithm 2 Equivalent SKG

```

for  $i = 1, \dots, E$  do
  //Select source node  $u$ 
   $u = 0$ 
  for  $j = 1, \dots, k$  do
    With probability  $U_r$  choose subregion  $r$ 
     $u = nu + r$ 
  end for
  //Select destination node  $v$ 
   $v = 0; z = u$ 
  for  $j = 1, \dots, k$  do
     $l = \text{mod}(z, n)$ 
    With probability  $V_{ls}$  choose subregion  $s$ 
     $v = nv + s; z = z/n$  (integer division)
  end for
  Add edge  $(u, v)$ 
end for

```

Here we note that the source node selection procedure is (approximately) a multinomial random variable with parameters E and $U^{[k]}$, where $U^{[k]}$ is the k -th Kronecker power of U .

2.2 Poisson Stochastic Kronecker Graph

Due to the following elementary result [4] we can replace the source node selection procedure, a multinomial random variable, with i.i.d. Poisson random variables.

Lemma. *Let X_1, \dots, X_s be independent Poisson random variables each with mean $\alpha p_1, \dots, \alpha p_s$, where $\alpha > 0$, $p_1, \dots, p_s \geq 0$, and $\sum_{i=1}^s p_i = 1$. Then*

$$\mathbf{P} \left(X_1 = x_1, \dots, X_s = x_s \mid \sum_{i=1}^s X_i = m \right)$$

$$= \frac{m!}{x_1! \cdots x_s!} p_1^{x_1} \cdots p_s^{x_s}$$

i.e. conditioned on the sum X_1, \dots, X_s is distributed as a multinomial distribution.

We are finally ready to state the main algorithm of this paper. Let E be the expected number of total edges while P, U, V, k are defined the same as before.

Algorithm 3 PSKG

```

Scatter  $E, P, U, V, k$ 
for Each vertex  $u$  do
  //Determine out-degree of  $u$ 
   $p = 1; z = u$ 
  for  $j = 1, \dots, k$  do
     $l = \text{mod}(z, n); p = pU_l$ 
     $z = z/n$  (integer division)
  end for
  Generate  $X \sim \text{Poisson}(Ep)$ 
  //For each edge determine destination vertex
  for  $i = 1, \dots, X$  do
     $v = 0; z = u$ 
    for  $j = 1, \dots, k$  do
       $l = \text{mod}(z, n)$ 
      With probability  $V_{ls}$  choose subregion  $s$ 
       $v = nv + s; z = z/n$  (integer division)
    end for
    Add edge  $(u, v)$ 
  end for
end for

```

PSKG will retain all the desired properties of SKG graphs. Specifically, say there is a desired property observed by SKG graphs of all sizes with probability $1 - \varepsilon$. Then the Poisson SKG graphs will also have the desired property with probability $1 - \varepsilon$ by the following lemma.

Lemma. *Let A be an event that occurs with probability $1 - \varepsilon$ for SKG graphs of all sizes. Then A will also hold with probability $1 - \varepsilon$ for Poisson SKG graphs as well.*

Proof.

$$\begin{aligned}
\mathbf{P}_{\text{Poisson}}(A) &= \mathbf{E} \left[\mathbf{P} \left(A \mid \sum_{i=1}^k X_i = m \right) \right] \\
&= \mathbf{E} [\mathbf{P}_{\text{Multinomial}}(A|m)] > \mathbf{E}[1 - \varepsilon] = 1 - \varepsilon
\end{aligned}$$

□

2.3 Probabilistic Load Balancing

In vertex-centric BSP, where each computational worker takes ownership to vertices and their outgoing edges, it is not a priori clear how to distribute them; some vertices have more neighbors than others so assigning an equal number to each worker will likely result in load imbalance. As the storage requirement of the graph structure is proportional to the number of neighbors, we shall discuss load balancing with the goal of distributing the number of edges equally.

Let N_w denote the total number of workers to balance the load among and $w_{\text{id}} = 0, 1, \dots, (N_w - 1)$ denote the individual processor number. Let $U^{[k]}$ be the k -th Kronecker power of U and $(U^{[k]})_u$ the expected load proportion for vertex u . Now we split the set of vertices into contiguous partitions (contiguous by node numbering) so that each partition has a total load of about $1/N_w$ and is owned by one processor. This procedure is illustrated in Figure 2.

To do this partitioning efficiently, however, one must avoid explicitly forming $U^{[k]}$. Algorithm 4² achieves this by traversing the decision tree without explicitly forming it.

One legitimate concern of this strategy is that the load is only balanced in expectation and therefore it is possible that with bad luck the actual load is highly unbalanced. However, Theorem 1 tells us that with high probability the load imbalance is small.

Theorem 1. *The α -level confidence interval of the maximum load over all computational nodes is $[\frac{E}{N_w}, \frac{E}{N_w} + \delta]$ where*

$$\delta = \sqrt{\frac{2E}{N_w}} \sqrt{\log N_w + |\log |\log(1 - \alpha)||}$$

where $\frac{E}{N_w}$ is the load under perfect balance. We can interpret δ as the degree of load imbalance.

Proof. We first make the assumption that the expected load, E , is split and distributed perfectly, i.e. each worker will have a load of X_i where X_0, \dots, X_{N_w-1} are i.i.d. Poisson random variables with mean E/N_w . Let M be the upper bound of

²The algorithm is a simplified version specifically for $n = 2$. The generalization to arbitrary n is straightforward.

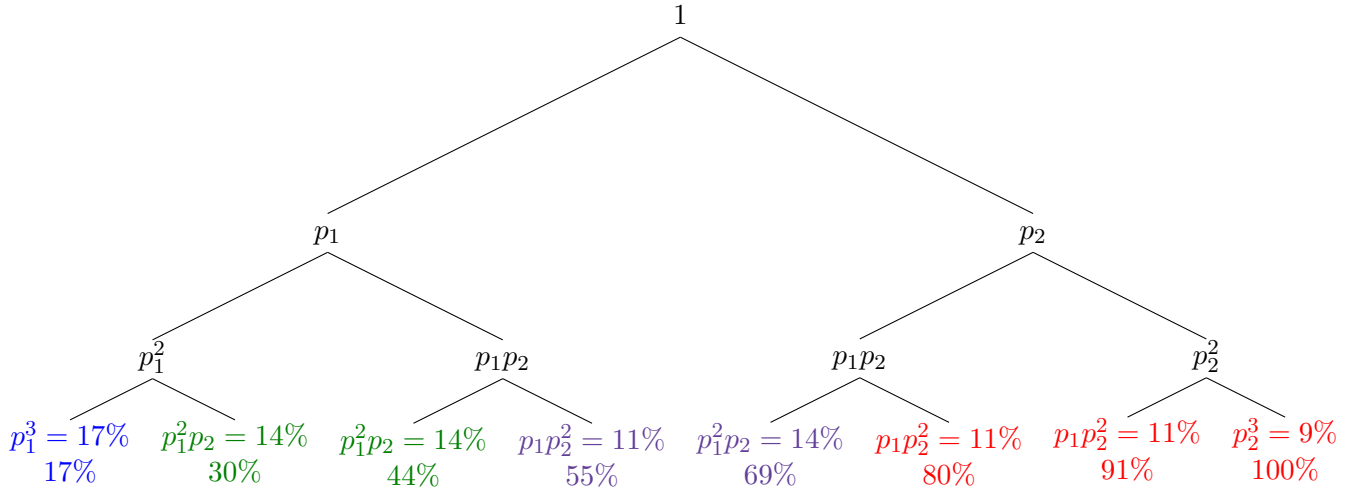


Figure 2: An example of load balancing where $n = 2$, $p_1 = 0.55$, $p_2 = 0.45$, $k = 3$, and $N_w = 4$. The second to last line denotes the load of each vertex and the last line denotes the cumulative load. The 4 processors attempt to take 25% of the total load each. Consequently processor number 0, 1, 2, and 3 takes ownership to the blue, green, purple, and red vertices, respectively.

the confidence interval.

$$\begin{aligned}
\mathbf{P} \left(\max_i X_i \leq M \right) &= \mathbf{P} (X_1 \leq M)^{N_w} \\
&= F_X^{N_w}(M) \leq 1 - \alpha \\
M &\leq F_X^{-1} \left((1 - \alpha)^{1/N_w} \right) \approx F_X^{-1} \left(1 + \frac{\log(1 - \alpha)}{N_w} \right) \\
&\approx \sqrt{\frac{E}{N_w}} \Phi^{-1} \left(1 + \frac{\log(1 - \alpha)}{N_w} \right) + \frac{E}{N_w}
\end{aligned}$$

$(1 - \alpha)^{1/N_w}$ is approximated by its Taylor series and X is approximated by a normal random variable $\mathcal{N}(E/N_w, E/N_w)$ given by the central limit theorem. Abramowitz[1] provides the following bound.

$$\sqrt{2\pi}(1 - \Phi(x)) = \int_x^\infty e^{-t^2/2} dt \leq 2e^{-x^2/2} \quad \text{for } x \geq 0$$

Using the fact that for non-increasing functions $f \leq g$ implies $f^{-1} \leq g^{-1}$ we arrive at the following.

$$\Phi^{-1}(1 - x) \leq \sqrt{2 \log x}$$

Putting these results together gives the theorem \square

Algorithm 4 Load Balancing

```

for Each worker do
   $r_{\text{low}} = w_{\text{id}}/N_w$ ;  $r_{\text{up}} = (w_{\text{id}} + 1)/N_w$ 
   $b = 0$ ; //lower bound
   $p_{\text{range}} = 1$ ; //probability range
   $u_{\text{low}} = 0$  //lower vertex id
  for  $i = 1, \dots, k$  do
    if  $r_{\text{low}} \leq b + pp_{\text{range}}$  then
       $p_{\text{range}} = pp_{\text{range}}$ ;  $u_{\text{low}} = 2u_{\text{low}}$ 
    else
       $b = b + pp_{\text{range}}$ 
       $p_{\text{range}} = (1 - p)p_{\text{range}}$ ;  $u_{\text{low}} = 2u_{\text{low}} + 1$ 
    end if
  end for
  Repeat above with  $u_{\text{up}}$ 
  Claim ownership to nodes  $u_{\text{low}}$  to  $u_{\text{up}}$ 
end for

```

3 Experimental Results

In this section, we demonstrate that PSKG and SKG generate graphs with essentially the same properties. As SKG models real world networks well [7]

the equivalence between PSKG and SKG implies the modeling power of PSKG.

To generate and analyze the SKG and PSKG graphs the SNAP library [10] and our own implementation of PSKG on Apache Giraph [6] were used, respectively.

3.1 Graph Patterns

There are several standard graph patterns that are used to compare the similarity between networks. In this paper, we shall use the following patterns: degree distribution, hop plot, scree plot, and network values. These choices are motivated by Leskovec’s [7] work.

Degree distribution: The histogram of the nodes’ degrees with exponential binning.

Hop plot: Number of reachable pairs $r(h)$ within h hops, as a function of the number of hops h .

Scree plot: Singular values of the graph adjacency matrix versus their rank.

Network values: Distribution of the principal eigenvector components versus their rank.

Figure 3 and Figure 4 compares the graph patterns of SKG and PSKG. It is clear that the results are essentially the same.

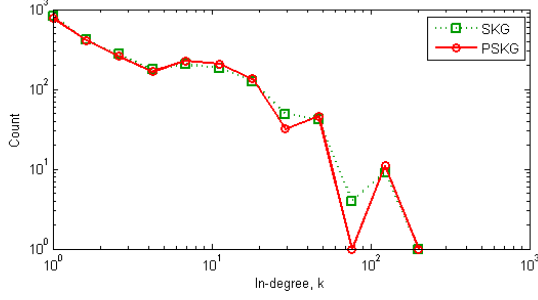
4 Conclusion

In conclusion, PSKG is a network generation model that is more efficient than and yet as powerful as SKG. Section 2 and 3 each provide theoretical and empirical evidence to this statement.

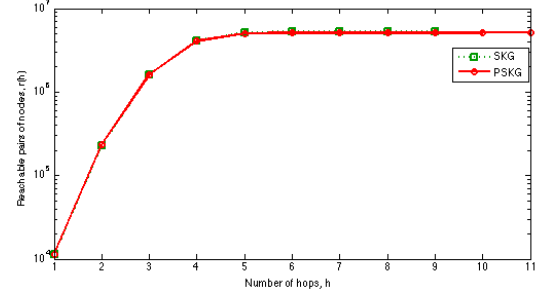
One promising direction of future work is vertex-centric algorithms for model estimation. There has been much work on SKG model fitting, which should directly apply to PSKG, but most do not concern vertex-centric parallelism. It would be interesting to see the efficiency a vertex-centric distributed fitting algorithm can achieve compared to a serial or MapReduce implementation.

References

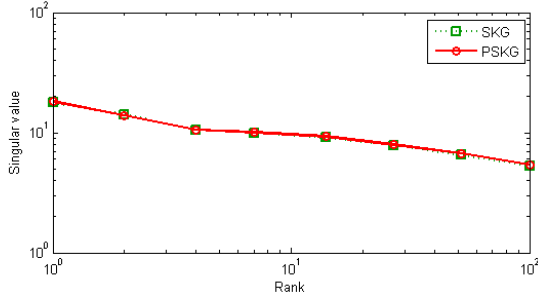
- [1] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications, 1964. ISBN: 0-486-61272-4.
- [2] *Apache Giraph*. URL: <http://giraph.apache.org/>.
- [3] *Apache Hama*. URL: <http://hama.apache.org/>.
- [4] Morton B. Brown and Judith Bromberg. “An Efficient Two-Stage Procedure for Generating Random Variates from the Multinomial Distribution”. English. In: *The American Statistician* 38.3 (1984), pp. 216–219. ISSN: 00031305.
- [5] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. “R-MAT: A recursive model for graph mining”. In: *In SDM*. 2004.
- [6] *Giraph Patch on Creating Randomized Synthetic Graph*. URL: <https://issues.apache.org/jira/browse/GIRAPH-26>.
- [7] Jure Leskovec et al. “Kronecker Graphs: An Approach to Modeling Networks”. In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 985–1042. ISSN: 1532-4435.
- [8] Grzegorz Malewicz et al. “Pregel: a system for large-scale graph processing”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. SIGMOD ’10. Indianapolis, Indiana, USA: ACM, 2010, pp. 135–146. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807184.
- [9] Semih Salihoglu and Jennifer Widom. *GPS: A Graph Processing System*. Technical Report. Stanford University.
- [10] *SNAP Network Analysis Library*. URL: <http://snap.stanford.edu/>.



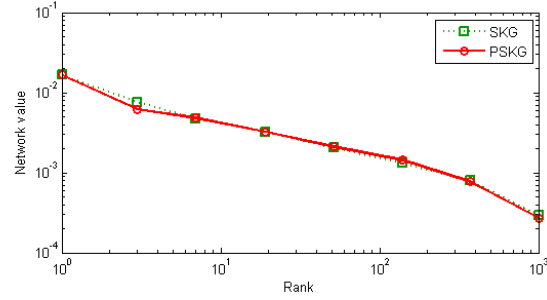
(a) Degree distribution



(b) Hop plot

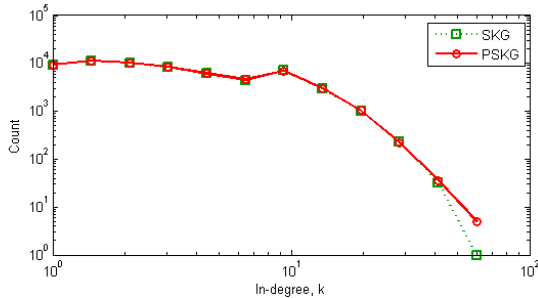


(c) Scree plot

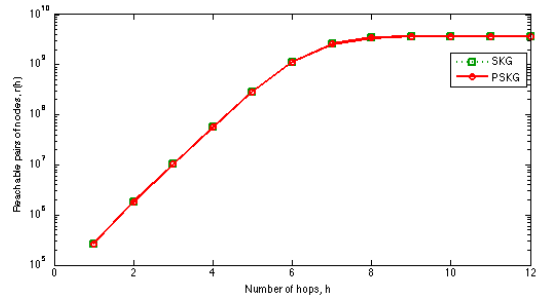


(d) "Network value"

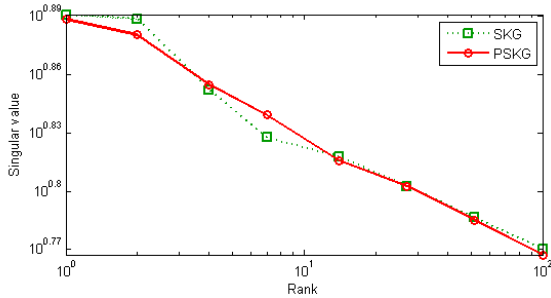
Figure 3: Graph patterns with parameters $n = 2$, $k = 12$, $E = 11400$, and $P = [0.4532, 0.2622; 0.2622, 0.0225]$



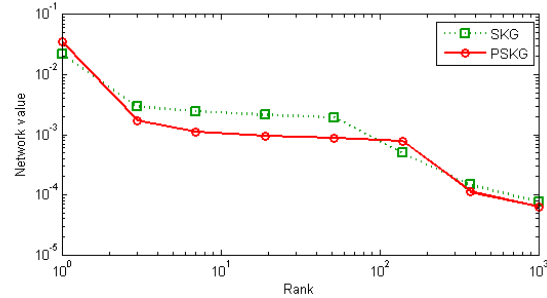
(a) Degree distribution



(b) Hop plot



(c) Scree plot



(d) "Network value"

Figure 4: Graph patterns with parameters $n = 4$, $k = 8$, $E = 263546$, and $P = [\alpha, \alpha, \alpha, \alpha; \alpha, \alpha, \beta, \beta; \alpha, \beta, \alpha, \beta; \alpha, \beta, \beta, \alpha]$ where $\alpha = 0.0861$ and $\beta = 0.0231$. P is the adjacency matrix of a star graph on 4 nodes (center + 3 satellites) with the 1's are replaced with α and the 0's are replaced with β .